Hybrid Cross Approximation and Shared-Memory Programming for the Electric Field Integral Equation

Priscillia Daquin[1], Jean-René Poirier[1], Ronan Perrussel[1], and Alfredo Buttari[2]
[1]Université de Toulouse, CNRS, LAPLACE, Toulouse 31071, France, firstname.name@laplace.univ-tlse.fr
[2]Université de Toulouse, CNRS, IRIT, Toulouse 31000, France, alfredo.buttari@enseeiht.fr

The Boundary Element Method (BEM) computation of the scattering of an electromagnetic wave can be accelerated using the hierarchical matrix format ($\mathcal{H}$-matrix). The matrix blocks representing the far interactions are approximated using the Hybrid Cross Approximation (HCA). It is then possible to perform an iterative solution method. The shared-memory parallelism of some operations will help to further optimize the total solution time of the scattering problem.

***Index Terms*—Electric field integral equation (EFIE), $\mathcal{H}$-matrix, hybrid cross approximation (HCA), parallel computing.**

I. Introduction

SCATTERING problems can be modeled by integral equations, such as the Electric Field Integral Equation (EFIE) [1], using the Boundary Element Method (BEM). Their discretization leads to a dense linear system

$$Z\boldsymbol{j} = \boldsymbol{e} \tag{1}$$

with $Z \in \mathbb{C}^{n\times n}$ defined as

$$Z_{ij} = \int_{\Gamma\times\Gamma} G(\boldsymbol{x},\boldsymbol{y})\Big(\boldsymbol{\varphi}_i(\boldsymbol{y})\cdot\boldsymbol{\varphi}_j(\boldsymbol{x}) - \frac{1}{k_0^2}\boldsymbol{\nabla}_\Gamma\cdot\boldsymbol{\varphi}_i(\boldsymbol{y})\boldsymbol{\nabla}_\Gamma\cdot\boldsymbol{\varphi}_j(\boldsymbol{x})\Big)\,\mathrm{d}\boldsymbol{x}\mathrm{d}\boldsymbol{y}, \tag{2}$$

and $\boldsymbol{e} = (e_k)_{1\le k\le n}$ defined as

$$\forall k\in\{1,...,n\}\quad e_k = \frac{\mathrm{i}}{k_0 Z_0}\int_\Gamma \boldsymbol{E}^{\text{inc}}(\boldsymbol{x})\cdot\boldsymbol{\varphi}_k(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x}, \tag{3}$$

where n is the number of Degrees Of Freedom (DOFs), $\boldsymbol{\varphi}_i$ the i^{th} Galerkin basis function, $\boldsymbol{\nabla}_\Gamma$ the surface gradient, k_0 the wave number, Z_0 the impedance of free space and G the Green kernel defined as

$$G(\boldsymbol{x},\boldsymbol{y}) = \frac{\mathrm{e}^{-\mathrm{i}k_0|\boldsymbol{x}-\boldsymbol{y}|}}{|\boldsymbol{x}-\boldsymbol{y}|}. \tag{4}$$

The *a priori* complexity of such a calculation is proportional to n^2 and restricts the BEM to relatively coarse grids. Hence we need to use a method to improve this complexity.

In this paper, we consider the $\mathcal{H}$-matrix format obtained by performing a Hybrid Cross Approximation (HCA). We then evaluate the optimization achieved thanks to the shared-memory parallel programming of the $\mathcal{H}$-matrix-vector product, which will lead to an improved iterative solver.

II. $\mathcal{H}$-matrix

A. Clustering

The $\mathcal{H}$-matrix term denotes a data-sparse matrix format that enables the memory storage and the complexity of the arithmetic operations, as the matrix-vector product, to scale as $n\log(n)$ instead of n^2. Thus it allows for solving large-scale problems efficiently using integral equation methods [2].

A $\mathcal{H}$-matrix is related to a hierarchy defined by a clustering technique of the DOFs. This clustering is performed by recursive bisections of the bounding boxes $\mathcal{B}$ containing the DOFs, until each box contains a maximum of $n_{\max}$ DOFs. This leads to a binary tree and a quadtree embodying the hierarchical structure of the $\mathcal{H}$-matrix (see Fig. 1).

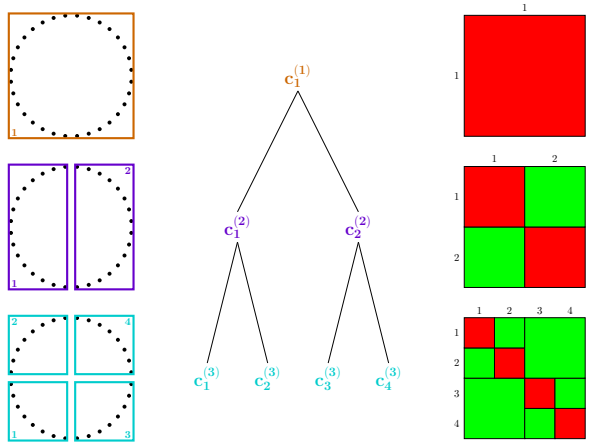

Fig. 1. Clustering of a circle discretized with 32 DOFs with $n_{max} = 8$ (3 hierarchical levels). Node $c_j^{(i)}$ is the index set for the j^{th} cluster of the i^{th} level. Block k,l represents the interactions between clusters k and l. The green blocks can be compressed unlike the red ones.

B. Low-rank matrices

Once a hierarchy is defined, we can compress the blocks corresponding to interactions between distant clusters. Let us denote by $\hat{r}$ and $\hat{c}$ the set of DOFs indices in two such distant clusters. The compression can be done by approximating directly $Z_{|\hat{r}\times\hat{c}}$, the submatrix of Z for the indices in $\hat{r}\times\hat{c}$, by a low-rank matrix computed by the Adaptive Cross Approximation (ACA) [2], or by replacing the kernel g by a degenerate $\widetilde{g}$ as it is done in the Hybrid Cross Approximation (HCA) [3] (see Section III). Both approaches lead to a low-rank approximation with a prescribed error ε as

$$Z_{|\hat{r}\times\hat{c}} \approx UV^H \tag{5}$$

where $Z_{|\hat{r}\times\hat{c}} \in \mathbb{C}^{m\times n}$, $U \in \mathbb{C}^{m\times k}$, $V \in \mathbb{C}^{n\times k}$, and k is the rank of the approximation.

C. Arithmetic of $\mathcal{H}$-matrices

We can develop an arithmetic of $\mathcal{H}$-matrices [4] including a $\mathcal{H}$-matrix-vector product, a rounded sum between two $\mathcal{H}$-matrices of same clustering structure, a formatted multiplication or a $\mathcal{H}$-LU decomposition.

Once the $\mathcal{H}$-matrix is assembled, we can also lower again the storage by using a coarsening technique which recursively agglomerates two or four blocks into one block and thus simplify the $\mathcal{H}$-matrix structure [2]. When used, this coarsening technique helps to reduce the computation time of the arithmetic operations without damaging the overall accuracy.

III. Hybrid Cross Approximation

The HCA [3] was first described in order to overcome some limitations of the ACA [2] concerning geometries with edges and kernels that contain a differential operator. This compression method has been applied to a magnetostatic formulation in [5], but is still rarely used today. We define the method for a kernel function g such as:

$$g(\boldsymbol{x},\boldsymbol{y}) = D_{\boldsymbol{x}} D_{\boldsymbol{y}} \gamma(\boldsymbol{x},\boldsymbol{y}) \tag{6}$$

where operators $D_{\boldsymbol{x}}$ and $D_{\boldsymbol{y}}$ are differential operators respectively according to $\boldsymbol{x}$ and $\boldsymbol{y}$, and γ an asymptotically smooth function, being well suited for the ACA computation.

The coefficients of Z built with the Galerkin method are defined for $i \in \{1,...,n_{\boldsymbol{x}}\}$ and $j \in \{1,...,n_{\boldsymbol{y}}\}$ as

$$Z_{ij} = \int_{\Gamma\times\Gamma} \boldsymbol{\varphi}_i(\boldsymbol{x}) g(\boldsymbol{x},\boldsymbol{y}) \boldsymbol{\psi}_j(\boldsymbol{y}) \,\mathrm{d}\boldsymbol{x}\mathrm{d}\boldsymbol{y} \tag{7}$$

The first step of the HCA consists in selecting some interpolation points $(\boldsymbol{x}_i)_{i\in I}$ and $(\boldsymbol{y}_j)_{j\in J}$ in the bounding boxes $\mathcal{B}_x$ and $\mathcal{B}_y$. Several approaches are available, all of them using the ACA algorithm. We chose to apply it on a tensorization of order m Chebychev interpolation points of $\mathcal{B}_x$ and $\mathcal{B}_y$.

Applying the ACA algorithm with a precision $\varepsilon_{\text{ACA}} = \varepsilon_{\text{HCA}}$ on those points provides two lists of pivot points $(\boldsymbol{x}_i)_{i\in I}$ and $(\boldsymbol{y}_j)_{j\in J}$, with $k = \#I = \#J$ the approximation rank. The degenerate kernel is $\widetilde{g}$ written as $\widetilde{g}(\boldsymbol{x},\boldsymbol{y}) = D_{\boldsymbol{x}} D_{\boldsymbol{y}} \widetilde{\gamma}(\boldsymbol{x},\boldsymbol{y})$ with

$$\widetilde{\gamma}(\boldsymbol{x},\boldsymbol{y}) = \begin{pmatrix} \gamma(\boldsymbol{x},\boldsymbol{y}_1) \\ \vdots \\ \gamma(\boldsymbol{x},\boldsymbol{y}_k) \end{pmatrix}^T M^{-1} \begin{pmatrix} \gamma(\boldsymbol{x}_1,\boldsymbol{y}) \\ \vdots \\ \gamma(\boldsymbol{x}_k,\boldsymbol{y}) \end{pmatrix} \tag{8}$$

and M defined as $m_{ij} = \gamma(\boldsymbol{x}_i,\boldsymbol{y}_j)$.

Defining matrices $A \in \mathbb{C}^{n_{\boldsymbol{x}}\times k}$ and $B \in \mathbb{C}^{n_{\boldsymbol{y}}\times k}$ such as

$$\begin{cases} A_{il} = \displaystyle\int_\Gamma \boldsymbol{\varphi}_i(\boldsymbol{x}) D_{\boldsymbol{x}} \gamma(\boldsymbol{x},\boldsymbol{y}_l) \,\mathrm{d}\boldsymbol{x} & (9) \\ B_{jm} = \displaystyle\int_\Gamma \boldsymbol{\psi}_j(\boldsymbol{y}) D_{\boldsymbol{y}} \gamma(\boldsymbol{x}_m,\boldsymbol{y}) \,\mathrm{d}\boldsymbol{y} & (10) \end{cases}$$

and factorizing $M^{-1} = C^T D$ leads to

$$\widetilde{Z}_{ij} = \sum_{p=1}^{k} \left(\sum_{l=1}^{k} C_{pl} A_{il} \right) \cdot \left(\sum_{m=1}^{k} D_{pm} B_{jm} \right). \tag{11}$$

Finally, $\widetilde{Z} = UV^H$ with $U = AC^T$ and $V^H = BD^T$.

In Table I some results are provided for different values of m and $\varepsilon_{\text{HCA}} = 1\times10^{-4}$ for a metallic sphere of radius $r = 1\,\text{m}$ with $n = 112206$ DOFs. The integrations are computed with 3 Gauss-Legendre quadrature points on each element. We evaluate the relative error on the bistatic Radar Cross Section compared to the analytical solution.

TABLE I
Numerical validation of the HCA.

m	2	3	4
Assembly time (s)	2621.15	4422.56	5005.74
Memory (GB)	9.42	7.20	6.97
Error	3.21×10^{-4}	9.94×10^{-5}	9.93×10^{-5}

IV. Shared-memory parallelism

The $\mathcal{H}$-matrix format is well suited to the use of a parallel computation on a shared memory architecture [6], as it is structured into independent blocks. It is thus possible to order the blocks corresponding to the leaves of the block cluster tree into a list of blocks. We apply a OpenMP parallel loop on this list, which will distribute the computation to the available threads. The parallel calculation allows for reducing the total computation time of some operations and helps to further optimize the solution time.

In order to validate the efficiency of the parallelization of the $\mathcal{H}$-matrix-vector product, we applied it 100 times for different values of n (the number of DOFs) and sharing it into 2, 4, 8 and 16 threads. Table II presents the results of this test.

TABLE II
Computation time of 100 $\mathcal{H}$-matrix-vector products.

n	Sequential	Parallel			
		2 threads	4 threads	8 threads	16 threads
4521	3.32 s	1.60 s	1.31 s	0.70 s	0.62 s
18 294	19.95 s	8.19 s	6.17 s	4.40 s	2.87 s
41 046	54.29 s	20.49 s	16.18 s	13.21 s	8.78 s
69 930	102.37 s	38.38 s	30.68 s	26.55 s	17.76 s

The parallel computation of the $\mathcal{H}$-matrix-vector product is faster than the sequential computation and shows some scalability up to 16 cores, considered that the $\mathcal{H}$-matrix-vector operation is memory bound. The work on this operation is still in progress and will allow for performing a shared-memory parallel iterative solver.

References

[1] Sadasiva M. Rao, Donald R. Wilton, and Allen W. Glisson. Electromagnetic Scattering by Surfaces of Arbitrary Shape. *IEEE Transactions on Antennas and Propagation*, 30:409–417, 1982.

[2] Mario Bebendorf. Approximation of boundary element matrices. *Numerische Mathematik*, 86:565–589, 2000.

[3] Steffen Börm and Lars Grasedyck. Hybrid cross approximation of integral operators. *Numerische Mathematik*, 101:221–249, 2005.

[4] Wolfgang Hackbusch. A Sparse Matrix Arithmetic Based on H-Matrices. - Part I: Introduction to H-Matrices. *Computing*, 62:89–108, 1999.

[5] Jonathan Siau, Olivier Chadebec, Ronan Perrussel, and Jean-René Poirier. Hybrid Cross Approximation for a Magnetostatic Formulation. *IEEE Transactions on Magnetics*, 51, 2015.

[6] Ronald Kriemann. Parallel h-matrix arithmetics on shared memory systems. *Computing*, 74:273–297, 2005.